HAAGA-HELIA
ammattikorkeakoulu

Front end -kehityksen tehostaminen

Case: Audile

Pekka Wallenius

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2014

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

20.11.2014

Tekijä(t)
Pekka Wallenius

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Front end -kehityksen tehostaminen
Case: Audile

Sivu- ja liitesivumäärä
24 + 3

Opinnäytetyön otsikko englanniksi
Improving Front End Development
Case: Audile

Opinnäytetyön aiheena on front end -kehityksen tehostaminen. Opinnäytetyössä tutkitaan front end -kehityksessä käytettävien työkalujen hyödyntämistä verkkokehityksessä. Tutkimus suoritettiin osana verkkosivustoprojektia, joka tehtiin Auditive Edu Oy:n toimeksiantona. Sivustoprojektissa kehitettiin Audile-yhtyeelle verkkosivusto.

Nykyään sovelluskehityksessä pyritään käyttämään hajautettua arkkitehtuuria, jossa verkkosivuston tai -palvelun arkkitehtuuri on jaettu käyttäjän selainpuolen ja palvelinpuolen sovellusosiin, eli front ja back endiin. Tässä opinnäytetyössä esitellään front end -kehitystä helpottavia työkaluja ja -menetelmiä ja niiden tehokkuutta arvioidaan case-osion kautta, jossa projektin vaatimusten mukaan on valittu siihen parhaiten soveltuvat työkalut.

Front end -kehityksen tueksi on tehty useita eri työkaluja, ja niitä löytyy kaikille front end -kehityksen tärkeimmille osa-alueille. Osa-aluekohtaisten työkalujen välillä erot ovat pieniä, joten olennaista työkalun valinnan kannalta on se, että löytää parhaiten itselle tai kehitystiimille sopivat työkalut. Työkalujen käyttö nopeuttaa ja selkeyttää kehitystä.

Koska kehitystä tehtiin yksin, ei työkalujen ja -menetelmien hyödyistä saada aivan täyttä hyötyä irti. Front end -kehityksen tueksi kehitetyt työkalut ovat parhaimmillaan kun kehitystä tehdään tiimissä.

Asiasanat
front end, verkkokehitys, esikäsittelijä, automatisointi

HAAGA-HELIA
University of Applied Sciences

Abstract

20.11.2014

Author(s)
Pekka Wallenius

Degree programme
Business Information Technology

Report/thesis title
Improving Front End Development
Case: Audile

Number of pages and appendix pages
24 + 3

This study reviews how to improve front end development. The primary objective of the study was to find the best tools and methods to achieve improvement in front end development. This study was made as a part of assignment to build a web site for Audile.

Nowadays web development often uses decoupled software architecture where different layers that build the system are interacting with each other. These layers are tightly dependent on each other so this way they can be developed independently. Typically in web development the architecture is split in to front end and back end. These tools are evaluated in the case part of the study with explanation why they have been chosen for the website project.

There are several different kinds of tools made to help with front end development. You can find tools for basically every different part of development process. Different tools usually have only minor differences, so the main point of choosing a tool is that it is the most suitable for the project. Using these tools and methods you can improve front end development process.

However, as this study dealt with a project that had only one developer the benefits were not extensive as they could have been with a development team.

Keywords
front end, web development, preprocessor, automation

Sisällys

1 Johdanto

Opinnäytetyö sai alkunsa Auditive Edu Oy:n toimeksiantona, tarkoituksena luoda Audile-bändille verkkosivusto palvelemaan kansainvälistä faniyleisöä ja levittämään tietoa yhtyeestä. Jo hyvin varhaisessa vaiheessa projektin toteutusta kävi ilmi, että keskittyminen itse kehitystyön tehostamiseen oli mielenkiintoisempi aihealue ja se mahdollisti myös paremmin uusien asioiden oppimisen projektin kehitystyön myötä.

Tässä opinnäytetyössä tarkastellaan front end -kehityksen tehostamiseen käytettäviä nykyaikaisia työkaluja ja -menetelmiä, sekä esitellään mitä näistä työkaluista valittiin Audile-sivustoprojektiin ja miksi. Aluksi käydään läpi front end -kehittäjän työkaluista olennaisin, eli tekstieditori. Tekstieditoreita löytyy paljon erilaisia, mutta tässä työssä esitellään ja vertaillaan niistä suosituimmat.

Front end -kehittäjille on tarjolla laaja valikoima erilaisia paketteja ja työkaluja, joita käyttämällä kehitystyötä voidaan nopeuttaa huomattavasti. Samalla kehitystyön ylläpidettävyys ja hahmotettavuus paranevat.

Esikäsittelijöillä pyritään selkeyttämään front end -kehityksessä käytettävien kielten (HTML, CSS, JavaScript) kirjoittamista. Opinnäytetyössä käydään läpi näille kielille kehitetyt suosituimmat esikäsittelijät ja esitellään niiden toimintaa.

Verkkokehityksen apuna käytetään tyypillisesti erilaisia kirjastoja, joilla helpotetaan kehitystä. Kirjastoilla saadaan vähemmällä työllä kehitettyä usein käytettyjä toiminnallisuuksia sivustolle. Tässä opinnäytetyössä käydään läpi muutaman esimerkin kautta, minkälaisia kirjastoja on olemassa ja miten kirjastoja, kehyksiä ja muita paketteja voidaan ylläpitää helposti front end -pakettienhallinnan avulla.

Opinnäytetyön case-osuudessa käsitellään projektin vaatimuksiin nojaten, mitä työkaluja projektissa käytettiin ja miksi, sekä mitä projektikohtaisia haasteita ilmeni.

Perimmäisenä tarkoituksena tässä opinnäytetyössä oli selvittää millaisilla työkaluilla front end -kehitystä voidaan tehostaa. Tämän lisäksi esitellään Audile WordPress-sivustoprojektin avulla, kuinka nämä työkalut soveltuvat käytännön kehitystyöhön ja minkälaisia etuja niillä saavutettiin.

2 Front end -kehitys

Nykyään verkkokehitystä voidaan usein ajatella hajautetuksi, jossa sivuston tai sovelluksen arkkitehtuurissa on eriytettynä käyttäjälle näkyvä, sekä palvelinpuolen sovellus. Yleisenä jaotteluna tässä käytetään front end- ja back end jaottelua. Front end -kehityksessä keskitytään käyttäjän selaimessa tapahtuviin toimintoihin ja käyttäjäkokemukseen, kun taas back end -kehitys pitää huolen sovelluksen tai sivuston toimintalogiikasta sekä tietojen tallennuksesta ja tarjoamisesta. Hajautettu arkkitehtuuri mahdollistaa esimerkiksi sovelluksen taustatoimintojen, eli back endin vaihtamisen teknologiasta toiseen ilman, että loppukäyttäjä havaitsee sovelluksen käytössä muutoksia.

Front end -kehitys on nopeasti kasvava ala, sillä projektit monimutkaistuvat jatkuvasti, ja aktiivinen kehittäjäyhteisö pyrkii löytämään näihin haasteisiin uusia ja tehokkaampia toimintatapoja. Front end -kehitys pitää sisällään loppukäyttäjälle näkyvien sivuelementtien ja niihin liittyvien vuorovaikutusten kehittämisen. Kehityksen lähtökohtana voidaan pitää sitä, että sivusto olisi helppokäyttöinen ja loppukäyttäjän saapuessa olennainen informaatio olisi helposti saatavilla.

Suurimpia haasteita kehittäjille aiheuttavat laajakirjo erilaisia päätelaitteita älytelevisioista mobiililaitteisiin. Laitteiden näyttöjen koko ja resoluutio monimutkaistavat kehitystä, kun sivuston tulisi toimia käyttöjärjestelmästä, selaimesta ja laitteesta riippumatta.

Front end -kehitys vaatii niin visuaalista silmää kuin myös hyvää ongelmanratkaisukykyä. Tarkoituksena on saada luotua loppukäyttäjää miellyttävä ja käyttökokemukseltaan loistava lopputulos. Sivuston elementit koostetaan HTML:llä, niiden ulkoasua voidaan hallita CSS-tyyleillä ja vuorovaikutteisen lopputuloksen aikaansaamiseksi käytetään JavaScriptiä.

3 Front end -kehityksen tehostaminen

Front end -kehityksen kohteena olevat verkkosivut ja -palvelut asettavat nykyaikana jatkuvasti enemmän toiminnallisia- ja visuaalisia vaatimuksia kehittäjille. Mobiililaitteet, suuret ja pienet näytöt hankaloittavat kehitystyötä merkittävästi. Kehityksessä käytettäviä kieliä ei alun perin suunniteltu käytettäväksi näin monimutkaisissa projekteissa, joten front end -kehityksessä joudutaan tukeutumaan yhä useammin kehitystä tukeviin ja helpottaviin työkaluihin.

Nykyaikana on onneksi saatavilla useita erilaisia työkaluja ja kirjastoja Front end -kehitystyötä tukemaan. Näistä suurin osa on avoimen lähdekoodin projekteja, joita ylläpidetään käyttäjäyhteisön toimesta.

3.1 Tekstieditorit

Tekstieditori on ehdottomasti verkkokehittäjän tärkein työkalu. Ilman minkäänlaista tekstieditoria kehitystyö ei yksinkertaisesti ole mahdollista. Oli kyseessä sitten komentoriviltä ajettava tekstieditori tai työpöytäsovellus, on se joka tapauksessa perimmäisenä vaatimuksena kehitystyössä. Erilaisia tekstieditoreita on olemassa lähes lukematon määrä, ja niitä tulee jatkuvasti lisää. Jotkut niistä ovat keskittyneet helppokäyttöisyyteen ja toisia on mahdollista muokata itselle sopivaksi lisäosien ja muiden asetusten avulla. Tekstieditoreita löytyy maksullisina ja ilmaisina versioina, myös avoimen lähdekoodin tekstieditoreita on olemassa. Yhteistä kaikille erilaisille tekstieditoreille on kuitenkin niiden nimensä mukaan tekstitiedostojen luominen ja muokkaaminen. (Pettit 2013.)

Tekstieditorien lisäksi tai niiden vaihtoehtona verkkokehityksessä käytetään usein myös ohjelmointiympäristöjä. Tyypillisesti ohjelmointiympäristöt pitävät sisällään tekstieditoreista tutut toiminnot, mutta tämän lisäksi myös laajan kirjon muita toimintoja. Ohjelmointiympäristöjen lähtökohtana on tarjota työkalut koko kehityskaaren ajalle tiedostojen muokkaamisesta ja luomisesta, kääntämiseen sekä virheidenpaikantamiseen ja korjaamiseen. Ohjelmointiympäristöt ovat tyypillisesti monimutkaisempia ja raskaampia tekstieditoreihin verrattuna. Ohjelmointiympäristöt soveltuvat paremmin back end -kehitykseen. Yksi suosituimmista ohjelmointiympäristöistä on alun perin Java-kehitykseen suunniteltu avoimen lähdekoodin Eclipse. Ohjelmointiympäristöt soveltuvat myös front end -kehitykseen, mutta kehittäjät suosivat yleensä tekstieditoreita ohjelmointiympäristöjen sijaan niiden keveyden, helppokäyttöisyyden ja joustavuuden ansiosta. (Pettit 2013.)

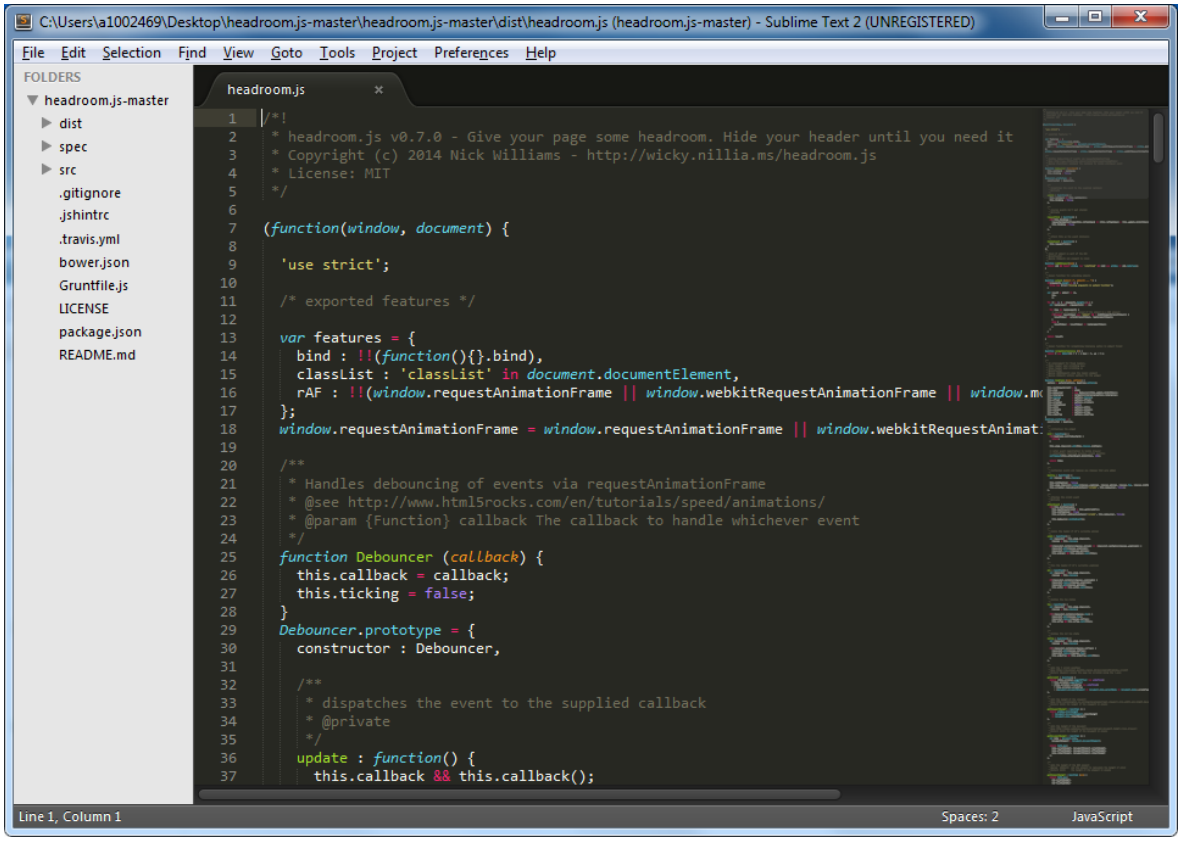
Tekstieditoreista suosituimpia ovat Brackets, Sublime Text sekä Atom. Brackets on Adoben alulle panema avoimen lähdekoodin tekstieditori, jonka kehitystä ylläpidetään GitHu-

bissa. Se on tekstieditori, joka on luotu HTML:llä, CSS:llä ja JavaScriptillä, eli kielillä, joita sillä on tarkoitus pääasiassa myös editoida. Brackets tukee Windows-, Linux- ja OS X-käyttöjärjestelmiä. Bracketsin vahvuuksia ovat avoin lähdekoodi ja reaaliaikainen koodin esikatselu. Editorissa tehtäviä muutoksia on mahdollista seurata selainikkunassa reaaliajassa. (Buckler 2014.)

Atom on GitHubin alullepanema avoimen lähdekoodin tekstieditori, joka on ilmainen ja tukee Bracketsin tapaan kaikkia merkittävimpiä käyttöjärjestelmiä. Atom integroituu vahvasti Gitiin ja GitHubiin, joten projektien versionhallinta onnistuu mainiosti suoraan editorista. Atomille on tarjolla useita erilaisia väriteemoja ja lisäosia, joilla tekstieditorin toimintaa voidaan laajentaa. Lisäosien lataaminen onnistuu helposti sisäänrakennetun pakettienhallinnan avulla. (Buckler 2014.)

Sublime Text edustaa Bracketsin ja Atomin tapaan tekstieditoreiden uutta sukupolvea, vaikka onkin vertailtavista tekstieditoreista vanhin. Voidaan sanoa, että Sublime Text on toiminut eräänlaisena suunnannäyttäjänä uusien tekstieditorien esiinnousussa. Sublime Text on kolmikosta ainoa maksullinen tekstieditori, mutta siitä on myös tarjolla kokeiluversio, jonka käyttöaikaa tai -ominaisuuksia ei ole rajattu. Ohjelma ilmoittaa satunnaisin väliajoin tallennuksien yhteydessä, että kyseessä on kokeiluversio ja tarjoaa mahdollisuutta ostaa lisenssin, mutta muutoin kokeiluversio vastaa täysin lisensoitua versiota. Myös Sublime Text tukee Windows-, Linux- ja OS X-käyttöjärjestelmiä. (Buckler 2014.)

Kaikki vertailtavat tekstieditorit ovat lähtökohtaisesti helppokäyttöisiä, joten ne soveltuvat hyvin niin aloitteleville, kuin jo kokeneemmille kehittäjille. Käyttöliittymältään kaikki tekstieditorit ovat hyvin samankaltaisia. Vasemmasta reunasta löytyy projektiin liittyvät tiedostot ja suurimman osan ruudusta vie itse tekstieditori. Sublime Text poikkeaa kahdesta muusta editorista siltä osin, että se tarjoaa ruudun oikeassa reunassa niin sanotun pienoiskartan avoimena olevasta koodista (kuva 1). Se antaa kokonaiskuvan koko koodista joka helpottaa koodissa liikkumista varsinkin isojen tiedostojen kohdalla. Pienoiskartta on mahdollista asentaa myös Bracketsiin lisäosana, mutta sen suorituskyky ei yllä Sublime Textin tasolle. Käyttöliittymiä voidaan muokata mieleiseksi joko valmiiden tai ladattavien väriteemojen avulla.



Kuva 1. Sublime Textin käyttöliittymä ja oikeassa reunassa näkyy pienoiskartta

Tekstieditorien toiminnallisuuksia voidaan laajentaa lisäosien avulla. Bracketsistä ja Atomista löytyy sisäänrakennettuna pakettienhallinta, jonka avulla lisäosien asentaminen on helppoa. Sublime Text vaatii paketinhallinnan asentamisen erikseen. Yksi suosituimmista lisäosista on Emmet, jonka tarkoituksena on nopeuttaa HTML:n ja CSS:n kirjoittamista, tarjoamalla laajennettavia lyhenteitä. Lyhenteet laajennetaan painamalla Tab-näppäintä. Lyhenteiden avulla voidaan luoda esimerkiksi hierarkkisia HTML-rakenteita (kuva 2).

```
// Emmet lyhenne

div>ul>li

// Lyhenne laajennettuna

<div>
    <ul>
        <li></li>
    </ul>
</div>
```

Kuva 2. Esimerkki Emmetin toimintalogiikasta

Tekstieditorien tehokkuuden puolesta puhuu myös mittava määrä pikanäppäin yhdistelmiä. Pikanäppäinten avulla kehitystä voidaan tietyissä tapauksissa nopeuttaa huomatta-

vasti. Pikanäppäinyhdistelmien avulla voidaan esimerkiksi muokata tiettyä merkkijonoa useassa paikassa samaan aikaan tai tehdä tehokkaita hakuja projektin tiedostoista.

Kaikki esitellyt tekstieditorit ovat loppujen lopuksi hyvin samankaltaisia, jokaiselta löytyy omat vahvuudet, mutta peruskäytön kannalta käyttökokemuksesta ja helppokäyttöisyydestä ei löydy merkittäviä eroja. Näin ollen tekstieditorin valinnassa on usein kyseessä tottumuskysymys. Pohjimmiltaan kaikki on kiinni siitä, mikä työkalu tuntuu itselle luontevimmalta käyttää ja mikä soveltuu omaan kehitystyönkulkuun parhaiten ja millä pystyy toimimaan mahdollisimman tehokkaasti.

3.2 Kehykset

Erilaisia kehyksiä on käytetty back end -ohjelmoinnin tukena jo vuosia, joten on myös luontevaa, että niitä löytyy myös front end -kehitykseen. Kehyksien käyttäminen kehityksen apuna ei ole pakollista, mutta ne ovat yhtenä työkaluna muiden joukossa. Niiden avulla saavutetut hyödyt puhuvat kuitenkin puolestaan. Kehyksen käyttämisen etuna front end -kehityksessä on se, että lähdekoodin rakenne pysyy vaivattomammin loogisena ja koodin luettavuus parantuu modulaarisuuden ja standardoitujen toimintatapojen ansiosta. (Symfony.)

Ylläpidettävyys ja päivitettävyys ovat myös kehysten vahvuuksia. Kehyksiä ylläpidetään yleensä avoimena lähdekoodina yhteisön voimin, jolloin kehitys ja vikojen korjaaminen eivät ole yksittäisen kehittäjän tai kehitystiimin vastuulla. Kehysten käyttäminen helpottaa sovelluksen tai sivuston jatkokehitystä, koska kehykset ohjaavat kehittäjää kirjoittamaan laadukasta ja yksiselitteistä koodia. (Symfony.)

Kehykset säästävät myös kehittäjien aikaa ja vaivaa, kun jokaista sivuston komponenttia ei tarvitse aina tehdä alusta asti. Kehyksen käyttäminen mahdollistaa siis sen, että kehityksessä voidaan keskittyä olennaiseen eli räätälöityjen ratkaisujen kehittämiseen. (Symfony.)

Front end kehityksessä kehykset voidaan jakaa karkeasti kahteen eri ryhmään; HTML ja CSS -kehyksiin ja JavaScript-kehyksiin. HTML ja CSS -kehyksistä suosituimpia ovat Twitterin kehittämä Bootstrap sekä ZURBin kehittämä Foundation. JavaScript-kehyksistä suosituimpia ovat AngularJS, React sekä Ember.js.

Bootstrapin pääasiallisena tarkoituksena on tarjota työkalut responsiivisen verkkosivuston luomiseen. Bootstrapin ennalta määrättyjen CSS-luokkien avulla responsiivisen sivuston

kehittäminen helpottuu huomattavasti, ja se tarjoaa samalla myös kattavan selaintuen sivustolle. Tämän lisäksi Bootstrap tarjoaa JavaScript-lisäosia ja valmiiksi määriteltyjä käyttöliittymäkomponentteja. Bootstrap-paketti on täysin räätälöitävissä ja onkin suositeltavaa, että Bootstrapia käytettäessä pakettiin on sisällytettynä vain sivustossa käytettävät ominaisuudet. Näin sivuston koko pysyy maltillisena ja se latautuu nopeammin. (Park 2013.)

3.3 Esikäsittelijät

Esikäsittelijät ovat ohjelmia jotka vastaanottavat tietyntyyppistä dataa ja prosessoivat sen toisenlaiseksi. Front end -kehityksessä joko HTML-, CSS- tai JavaScript-tiedostoiksi. Esikäsittelijät ovat ikään kuin kerros ylläpidettävien ja selaimelle tarjottavien tiedostojen välissä (Cederholm 2013). Niiden käyttämä syntaksi vaihtelee, mutta lähtökohta kaikissa on sama. Tarkoituksena on tuottaa hyvin jäsenneltyä ja kaunista koodia, joka samalla helpottaa koodin kirjoittamista ja lukemista. Esikäsittelijöiden avulla pyritään myös välttämään samojen asioiden toistamista ja näin osaltaan tehostamaan kehitystä. (Howe.)

Esikäsittelijöiden käyttäminen antaa kehittäjälle selvän hyödyn, mutta kääntäjien käyttäminen voi osoittautua kokemattomille kehittäjille hankalaksi. Esikäsittelijöiden kääntämiseen käytetään yleensä komentorivityökaluja, mutta myös huomattavasti yksinkertaisempia työpöytäsovelluksia löytyy, jotka ajavat saman asian. Työpöytäsovellukset ovat yleensä maksullisia ja niitä voidaan käyttää vain tiettyjen esikäsittelijöiden kanssa. Suosituimpia näistä työpöytäsovelluksista ovat Prepos ja CodeKit. (Chouhan 2014.)

Esikäsittelijöiden hyödyntäminen projekteissa edellyttää suunnitelmallisuutta ja hyvää näkemystä tulevasta projektista. Näin pystytään valitsemaan projektiin parhaiten soveltuvat esikäsittelijät.

3.3.1 HTML-esikäsittelijät

Tyypillisesti HTML:n kirjoittaminen on itseään toistavaa ja jokainen elementti joudutaan sulkemaan erikseen. Vaikka nämä työvaiheet ovat pieniä, ne toistuvat usein ja näin ollen hidastavat HTML-koodin kirjoittamista. Tämän kaltaisten ongelmien ratkaisuksi on kehitetty HTML-esikäsittelijöitä. Näistä suosituimpia ovat Markdown, Haml, ja Jade. (Howe.)

Haml toimii hyvänä esimerkkinä siitä, miten HTML-esikäsittelijällä voidaan selkeyttää HTML:n kirjoittamista. Hamlissa elementtejä ei tarvitse erikseen sulkea, vaan ne erotetaan toisistaan rivin vaihdoilla. Sisennykset vastaavat HTML-rakenteen hierarkiaa (kuva 3).

Haml hyväksyy myös perinteisen HTML-syntaksin koodin ja joissain tapauksissa HTML:n kirjoittaminen saattaa olla jopa yksinkertaisempaa kuin Haml-vastineen. (Johnson 2012.)

```
// Haml
#container
  .box
    %h2 Jokin otsikko
    %p Jotain tekstiä
    %ul.mainList
      %li Ensimmäinen
      %li Toinen
      %li Kolmas

// HTML vastine
<div id="container">
  <div class="box">
    <h2>Jokin otsikko</h2>
    <p>Jotain tekstiä</p>
    <ul class="mainList">
      <li>Ensimmäinen</li>
      <li>Toinen</li>
      <li>Kolmas</li>
    </ul>
  </div>
</div>
```

Kuva 3. Esimerkki Haml-syntaksista ja sama käännettynä HTML:ksi

3.3.2 CSS-esikäsittelijät

Suosituimpia CSS-esikäsittelijöitä ovat Sass, LESS ja Stylus. CSS-esikäsittelijät mahdollistavat esimerkiksi muuttujien, sisäkkäisten sääntöjen sekä matemaattisten operaattorien hyödyntämisen, joka tavallisesti ei olisi CSS:n kanssa mahdollista. Nämä uudet ominaisuudet helpottavat tyylitiedostojen ylläpitoa. (Cederholm 2013.)

Lähtökohtaisesti CSS:n suurena ongelmana on se, että sitä ei alun perin suunniteltu niin monimutkaisiin käyttötarkoituksiin kuin mihin sitä nykyään käytetään. Tähän ongelmaan CSS-esikäsittelijät pyrkivät löytämään toimivan ratkaisun. (Cederholm 2013.)

CSS:n ongelmista huolimatta CSS-esikäsittelijöiden käyttöönottoa saatetaan vältellä niiden näennäisen monimutkaisuuden takia, mutta esimerkiksi Sass käyttää samaa syntaksia CSS:n kanssa tarjoten siihen uusia ominaisuuksia. Sass-tiedostot käännetään CSS-tiedostoiksi. (Cederholm 2013.)

Sassissa muuttujat määritellään $-etuliitteellä. Muuttujat ovat hyödyllisiä kun määritellään arvoja, joita käytetään useammassa kuin yhdessä paikassa. Tällaisia arvoja ovat esimerkiksi värit ja fonttikoot, mutta muuttujia voidaan hyödyntää myös minkä tahansa muun

arvon tilalla. Esimerkkinä voidaan ottaa tilanne, jossa muuttujan ”site-main-color” arvoksi on annettu esimerkiksi ”#e3e3e3”. Kun muuttujaa on käytetty määrittelemään erinäisten elementtien väriarvoja ja ne syystä tai toisesta halutaan vaihtaa, onnistuu se muuttamalla muuttujan arvoa yhdessä paikassa. Tämä helpottaa huomattavasti tyylitiedostojen ylläpitoa. (Gyllenswärd 2013.)

Yksi Sassin merkittävimmistä ominaisuuksista on mahdollisuus jakaa tyylitiedostot osiin. Sass mahdollistaa tiedostojen sisällyttämisen tiedostoon käyttämällä @import-direktiiviä (kuva 4). Tyylitiedostojen jakaminen osiin ei vaikuta suorituskykyyn, koska ne käännetään kuitenkin ennen käyttöä yhdeksi CSS-tiedostoksi. Sassin kanssa käytetään tyypillisesti yhtä päätiedostoa, johon muut tiedostot sisällytetään. Sisällytettävät tiedostot ovat yleensä nimetty niin, että ne alkavat _-merkillä. Sass ei suoraan ota kantaa siihen, miten tyylitiedostojen jaottelun pitäisi tapahtua, mutta projektista riippuen kannattaa hyödyntää hakemistorakenteita ja jaotella esimerkiksi apu- ja komponenttikohtaiset tiedostot omiin kansioihin. (Giraudel 2014.)

```
// Core variables and mixins
@import "bootstrap/variables";
@import "bootstrap/mixins";

// Reset and dependencies
@import "bootstrap/normalize";
@import "bootstrap/print";
@import "bootstrap/glyphicons";

// Core CSS
@import "bootstrap/scaffolding";
@import "bootstrap/type";
@import "bootstrap/code";
@import "bootstrap/grid";
@import "bootstrap/tables";
@import "bootstrap/forms";
@import "bootstrap/buttons";
```

Kuva 4. Esimerkki Sass-tiedostosta, joka sisällyttää muita tiedostoja

Sass mahdollistaa myös sisäkkäisten sääntöjen kirjoittamisen. Tämä helpottaa yksityiskohtaisempien valitsijoiden kirjoittamisen ilman toistoa (kuva 5). Sisäkkäiset säännöt hyväksyvät myös media kyselyjen käytön, jolloin responsiivista sivustoa kehittäessä kaikki tietyn elementin säännöt sijaitsevat yhdessä paikassa. Myös tämä omalta osaltaan helpottaa tyylitiedostojen ylläpitoa ja parantaa luettavuutta. (Wendell 2012.)

```
// Sass
#navbar {
  width: 80%;
  height: 25px;

  ul { list-style: none; }

  li {
    float: left;
    a { text-decoration: none; }
    &:hover { text-decoration: underline; }
  }
}

// käännetty CSS
#navbar {width: 80%; height: 25px;}
#navbar ul {list-style: none;}
#navbar li {float: left;}
#navbar li a {text-decoration: none;}
#navbar li a:hover {text-decoration: underline;}
```

Kuva 5. Esimerkki Sass-syntaksista ja käännetystä CSS:stä

3.3.3 JavaScript-esikäsittelijät

Kuten HTML- ja CSS-esikäsittelijöitä, myös JavaScript-esikäsittelijöitä löytyy useampi kuin yksi. Niistä suosituimmat ovat CoffeeScript ja LiveScript. LiveScript on epäsuorajälkeläinen CoffeeScriptille, joten niistä löytyy paljon yhteneväisyyksiä. Ne eivät varsinaisesti tarjoa JavaScriptiin verrattuna mitään uutta, vaan niiden toiminnan perustana on tehdä JavaScriptin kirjoittamisesta helpompaa niin, että koodi pysyy myös luettavampana. JavaScript-esikäsittelijät kääntävät lähdetiedoston aina JavaScriptiksi. (Shull 2014.)

Molempien esikäsittelijöiden syntaksit ovat saaneet vaikutteita Python-, Ruby- ja Haskell-kielistä. JavaScriptistä, ja monista muista kielistä, tutuille aaltosulkeille ja puolipisteille ei ole tarvetta, vaan ne on korvattu rivinvaihdoilla ja sisennyksillä. Tämä pakottaa kehittäjän kirjoittamaan koodia selkeällä ja jäsennellyllä tavalla, jolloin muiden kehittäjien on helpompi lukea sitä. JavaScript-esikäsittelijöiden suurimpana heikkoutena on virheiden etsiminen ja paikantaminen, koska esimerkiksi CoffeeScript-tiedoston koodi poikkeaa huomattavasti lopputuloksena saadusta JavaScript-koodista (kuva 6). (Graham 2014.)

```
// CoffeeScript esimerkki

for num in [1..5]
  if num % 2 == 0
    console.log "#{num} on parillinen"
  else
    console.log "#{num} on pariton"


// ja käännettynä JavaScriptiksi

var num, _i;

for (num = _i = 1; _i <= 5; num = ++_i) {
  if (num % 2 === 0) {
    console.log("" + num + " on parillinen");
  } else {
    console.log("" + num + " on pariton");
  }
}
```

Kuva 6. Esimerkki CoffeeScriptin syntaksista ja alla CoffeeScript JavaScriptiksi käännettynä

3.4 Kirjastot ja lisäosat

Verkkokehityksen apuna voidaan käyttää myös erilaisia kirjastoja ja lisäosia. Front end -kehityksessä kirjastot ovat tyypillisesti JavaScript-kirjastoja, yksi suosituimmista on ehdottomasti jQuery. Front end -kehityksessä käytettävät lisäosat ovat usein jQueryllä kirjoitettuja, eli ne vaativat jQueryn toimiakseen. Kirjastojen tarkoituksena on tarjota kehittäjälle valmiita funktioita, joiden avulla voidaan yksinkertaistaa kehitystä.

jQuery on selainpuolen JavaScript-kirjasto. Sen tärkeimpänä käyttökohteena on dokumenttioliomallin muokkaaminen, eli toisin sanoen sillä voidaan lisätä, poistaa tai muokata jo olemassa olevia HTML-elementtejä. Tämän lisäksi sillä voidaan tehdä AJAX-kutsuja ja animoida HTML-elementtejä. jQuery ei tarjoa sovelluskehystä, vaan se on työkalu, joka helpottaa JavaScriptin-käyttöä. jQuery käyttää CSS3:n valitsinmoottoria, joten elementtien valinta toimii kuten CSS:ssä. (Hansson 2012.)

Front end -kehityksessä tulee usein tilanne vastaan, kun kehitettävälle sivustolle tarvitaan esimerkiksi kuvakaruselli tai jokin muu JavaScriptiä hyödyntävä toiminnallisuus. Toiminnallisuus voi olla monimutkainen kokonaisuus tai hyvin pieni asia, mutta todennäköisesti

joku toinen kehittäjä on tarvinnut vastaavaa toiminnallisuutta omalle sivustolleen ja hän on samalla luonut tästä toiminnallisuudesta lisäosan ja laittanut sen esimerkiksi GitHubiin tarjolle. Tämän johtuen lisäosia on tarjolla kaikkiin tyypillisiin toiminnallisuuksiin, mitä sivustokehittämisessä tarvitaan. Lisäosia käyttämällä voidaan sivustolle saada haluttuja toiminnallisuuksia hyvin pienellä vaivalla ja helposti.

3.5 Front end pakettienhallinta

Kirjastoja, lisäosia ja erilaisia työkaluja on nykyään olemassa enemmän kuin koskaan aikaisemmin. Ei ole poikkeuksellista, että verkkokehitysprojektissa saattaa olla käytössä kymmeniä eri kolmansien osapuolien kehittämiä ratkaisuja. Projekti saattaa käyttää responsiivisuuden saavuttamiseksi Bootstrap-kehystä, JavaScript-kirjastona jQueryä, ja tämän lisäksi useita JavaScript-lisäosia esimerkiksi kuvakaruselleja varten. Näiden ylläpitäminen on tavanomaisesti ollut vaativaa ja aikaa vievää, kaikki paketit ovat olleet ylläpidettyinä eri paikoissa ja jokaisen paketin on joutunut päivittämään erikseen. Tähän ongelmaan Twitter kehitti vuonna 2012 ratkaisuksi Bowerin, pakettienhallinta työkalun front end -kehitystä varten. Bower toimii erinomaisesti myös dokumentoinnin tukena, kaikki front end -riippuvuudet löytyvät helposti yhdestä tiedostosta, bower.json. (West 2013, Bradley 2014.)

Bowerin tarkoituksena on hallita projektissa käytettäviä kehyksiä, kirjastoja, lisäosia, työkaluja ja oikeastaan mitä tahansa front end -kehitykseen liittyvää. Bower vaatii toimiakseen Node.js:n, Node.js pakettienhallinta työkalun ”npm”:n sekä Git-versionhallintatyökalun. (Bower a)

Bower voi asentaa paketteja, jotka voivat olla joko rekisteröityjä, kuten esimerkiksi ”jQuery”, tai ne voivat olla ylläpidettyinä muualla. Bowerin avulla onnistuu myös epäviralisten pakettien asentaminen, joita ylläpidetään esimerkiksi Git-versionhallinnassa tai jossain tietyssä URL-osoitteessa. Bowerin rekisteröityjä paketteja voidaan etsiä Bowerin kotisivuilta tai komentoriviltä (kuva 7).

```
walpek@walpek-MS-7733:~$ bower search owlcarousel
Search results:

    owlcarousel git://github.com/OwlFonk/OwlCarousel.git
    OwlCarousel git://github.com/OwlFonk/OwlCarousel.git
    owl.carousel git://github.com/OwlFonk/OwlCarousel2.git
    owl-carousel git://github.com/OwlFonk/OwlCarousel.git
    owl-carousel2 git://github.com/OwlFonk/OwlCarousel2.git
    my-owl-carousel git://github.com/antsyd/OwlCarousel.git
    lazarofl.owlcarousel git://github.com/lazarofl/OwlCarousel.git
walpek@walpek-MS-7733:~$
```

Kuva 7. Esimerkki Bowerilla komentoriviltä tehdyn haun tuloksista

Uuden paketin asentaminen projektiin tapahtuu ”bower install”-komennolla. Tämä komento vaatii parametriksi asennettavan paketin nimen tai sijainnin. Uutta pakettia lisätessä on suositeltavaa myös käyttää ”–save-dev” -parametria. Näin uusi paketti lisätään bower.json -tiedostoon projektin riippuvuudeksi, helpottamaan ylläpitoa. ”bower install” -komennon yhteydessä on mahdollista myös määritellä asennettavan paketin versionumero alla olevan esimerkin mukaisesti. Bower hyväksyy versionumeroiksi esimerkiksi versionumero-alueen, Git-tagin tai Git-haaran nimen. (West 2013.)

```
bower install <paketti>#<versionumero>
```

Bower tallentaa oletuksena kaikki paketit ”bower-components” -hakemistoon. Tämä hakemisto luodaan samaan hakemistoon kuin missä bower-komento on suoritettu. Uusi oletushakemisto voidaan määritellä .bowerrc konfiguraatiotiedostoon, mikäli se halutaan muuttaa joksikin muuksi. (West 2013.)

Projektissa bower.json-tiedostoon määriteltyjen riippuvuuksien päivittäminen onnistuu ”bower update” -komennolla. Jos komennon perään ei ole määritelty minkään paketin nimeä, käy Bower bower.json-tiedoston läpi ja päivittää sen ohjeistuksen mukaan tarvittavat paketit. ”bower list” -komennolla voidaan tarkastaa mitä riippuvuuksia kyseisessä projektissa on (kuva 8). (Bradley 2014.)

```
walpek@walpek-MS-7733:~/projects/audile-wp/wp-content/themes/audile$ bower list
bower check-new     Checking for new versions of the project dependencies..
audile-wordpress#0.0.1 /home/walpek/projects/audile-wp/wp-content/themes/audile
├── CMB2#e52d7888b2
├── OwlCarousel#1.3.2
├─┬ bootstrap-sass-official#3.2.0+2
│ └── jquery#2.1.1
├── font-awesome#4.2.0
├── instafeed.js#1.3.2
└── twitter-fetcher#71fb5f1b69
```

Kuva 8. Projektin Bower-riippuvuudet saadaan listattua ”bower list” komennolla

Koska Bower-paketteja tarvitaan vain projektin rakennus- tai käännösvaiheessa, ei niitä tyypillisesti säilytetä versionhallinnassa. Tähän on olemassa kuitenkin eriäviä mielipiteitä. Pakettien sisällyttämistä versionhallintaan puoltaa esimerkiksi se, että mikäli paketin alkuperäinen sijainti syystä tai toisesta katoaa, on projektin kannalta tärkeä paketti kuitenkin vielä versionhallinnassa tallessa. Isojen projektien kohdalla tämä tarkoittaa yleensä versionhallinnan koon kasvua. Juuri tämän takia siirrettävissä projekteissa, pakettien sisällyttämistä versionhallintaa pidetään yleisesti parhaana käytäntönä. Lopulta tärkeää on kuitenkin se, että pakettien pois jättämiselle tai sisällyttämiselle on olemassa jokin perusteltu syy. (Osmani 2013.)

3.6 Automatisointi

Kehittämisen yhteydessä ilmenee jatkuvasti toistuvia tehtäviä, joita kehittäjä joutuu suorittamaan manuaalisesti. Suurimman osan näistä tehtävistä voi automatisoida, jolloin säästetään kehitykseen kulunutta aikaa, ja sitä myöten saavutetaan samalla taloudellisia säästöjä. Tyypillisesti automatisointia hoitaa JavaScript-tehtävänsuorittaja, mutta myös esikäsittelijät voivat automatisoida toimintaa. On kuitenkin suositeltavaa hoitaa tehtävienautomatisointi keskitetysti, jotta vältytään mahdollisilta ristiriidoilta ja siltä, että jouduttaisiin tekemään useampi toimenpide yhden sijaan. (Tucker 2013.)

Suosituimpia JavaScript-tehtävänsuorittajia ovat Grunt, Gulp ja Broccoli. JavaScript-tehtävänsuorittajia voidaan ajatella työnkulkutyökaluina. Tehtävänsuorittajia hyödyntäessä joudutaan etukäteen suunnittelemaan projektin työnkulkua, tarvittavia työkaluja ja projektin rakennetta. JavaScript-tehtävänsuorittajat mahdollistavat lähes kaikenlaisen työnkulun suorittamisen. Yksittäisiä tehtäviä voivat olla esimerkiksi tiedostojen ketjuttaminen yhdeksi tiedostoksi, tiedoston minimointi tai tiedoston poistaminen. Vaihtoehtoisesti tehtäviä voidaan myös kehittää omien tarpeiden mukaan. (Tucker 2013.)

4 Versionhallinta kehityksen tukena

Tavanomaisesti verkkokehitysprojektissa on mukana useampi kehittäjä, tällöin ongelmaksi muodostuu yleensä tiedostojen siirtelyn vaikeus ja niiden ajan tasalla pitäminen. Tiedostojen siirtely esimerkiksi sähköpostilla, FTP-ohjelmilla tai ylläpitäminen verkkolevyllä, ei yksinkertaisesti ole projektin kehitystyön kannalta järkevää tai tehokasta, koska tiedostoihin tehdyistä muutoksista on vaikea pysyä perillä tai ne saattavat vahingossa joutua ylikirjoitetuksi toisen kehittäjän toimesta.

Nykyisissä verkkokehitysprojekteissa yksi tärkeimmistä työkaluista on ehdottomasti versionhallintatyökalu. Versionhallinta toimii paitsi eräänlaisena varmuuskopiona, se myös helpottaa muutosten seuraamista. Jos vastaan tulee tilanne, että viimeisin muutos ei tuottanutkaan toivottua tulosta, voidaan versionhallinnan avulla esimerkiksi palata lähtötilanteeseen helposti. Versionhallinta mahdollistaa teoriassa rajattoman määrän kehittäjiä työstämään samaa projektia. (Stansberry 2008.)

Versionhallintatyökaluista suosituimpia ovat Subversion (SVN), Mercurial ja Git. Subversion on Apachen kehittämä CVS:n ideologiaa hyödyntävä keskitetty versionhallintatyökalu. Keskitetyllä versionhallinnalla tarkoitetaan sitä, että versionhallinnan tiedot sijaitsevat yhdessä paikassa keskitetysti. Mercurial ja Git ovat sen sijaan hajautettuja versionhallintatyökaluja, jossa jokaisella etävaraston käyttäjällä on omalla koneellaan täydellinen versio versionhallinnasta. Tämä mahdollistaa versionhallinnan käytön myös silloin kun internetyhteyttä ei ole saatavilla.

Git sai alkunsa kun Linus Torvaldsin kehittämä Linux kernel -projekti joutui vuonna 2005 luopumaan silloisesta versionhallintatyökalusta. Linus ja Linux-kehitysyhteisö rupesivat kehittämään versionhallintatyökalua, joka olisi nopeampi, yksinkertaisempi ja joka pystyisi paremmin hallitsemaan isompia projekteja, kuin aikaisemmin. Tähän kehitettiin Git-työkalu. (Git.)

Gitin suosiota on siivittänyt Linux-kehityksessä saadun huomion lisäksi myös palveluntarjoajien tuki. GitHub tarjoaa yksityisten- ja maksullisten etävarastojen lisäksi ilmaisia julkisia etävarastoja. Julkiset etävarastot soveltuvat täydellisesti avoimen lähdekoodin ohjelmille, joka on ollut yksi merkittävä tekijä Gitin suosiolle. Myös BitBucket tarjoaa maksullisten etävarastojen lisäksi ilmaisia palveluita. BitBucketista on mahdollista saada ilmaisia yksityisiä etävarastoja korkeintaan viiden käyttäjän tiimille. BitBucket tukee Gitin lisäksi Mercurialia. (Asay 2014.)

5 Case: Audile

5.1 Tavoite ja suunnitelma

Opinnäytetyön case-osuuden toimeksianto tuli Auditive Edu Oy:ltä, sen tavoitteena oli rakentaa Audile -yhtyeelle uusi sivusto palvelemaan kansainvälistä faniyleisöä. Sivuston kehityksessä keskityttiin rakentamaan visuaalisesti näyttävä ja responsiivinen sivusto. Sivustoprojektissa pyrittiin myös tuomaan esille yhtyeen sosiaalisen median kanavia mahdollisimman hyvin.

Opinnäytetyötä varten suunnittelin Audilen kanssa yhteistyössä, sivuston ulkoasun. Audilelta toiveina tulivat keskittyminen sosiaalisen median eri kanavien esille tuomiseen ja kuvamateriaalien näyttävään hyväksikäyttöön sekä sivuston mukautuminen saumattomasti myös mobiililaitteille.

Alun perin opinnäytetyön tarkoitus siis oli keskittyä esittelemään WordPress-sivuston kehityksen vaiheita, mutta jo hyvin varhaisessa vaiheessa kävi kuitenkin ilmi, että front end -kehityksen tehostaminen olisi kiinnostavampi ja ajankohtaisempi aihe kuin pelkkä sisällönhallintajärjestelmän päälle toteutettava sivusto. Uusi aihe mahdollisti myös sen, että opinnäytetyötä tehdessä pystyin oppimaan uusia asioita huomattavasti enemmän.

Vastatessani projektin kehityksestä yksin sain vapaat kädet projektinhallintaan. Projektin lopputuloksena kehitettävä sivusto oli tarkoitus julkaista kokonaisuudessaan Audilen uuden single-julkaisun ”Ride On” kanssa samanaikaisesti. Tämän takia päädyin käyttämään projektinhallinnassa ketterän Scrumin sijasta vesiputousmallia.

Projektin tavoitteena valita sivustonkehitykseen opinnäytetyössä esitellyistä erilaisista työkaluista ja -menetelmistä siihen parhaiten soveltuvat ja niiden avulla tehostaa ja nopeuttaa kehitysprosessia. Projektin tavoiteltuna lopputuloksena on konfiguraatiot käytetyistä front end -kirjastoista ja pakeista sekä projektin automaatiosta, kehitystyötä ja tuotantoon siirtoa varten.

5.2 Toteutus

Audile halusi pystyä päivittämään sisältöä vaivattomasti, joten sivuston sisällönhallintajärjestelmäksi valikoitui WordPress. Se toimi määräävänä tekijänä siinä, mitä työkaluja ja minkälaisia menetelmiä kehityksessä pystyttiin hyödyntämään.

Kehitystyötä tehtiin yhden henkilön voimin ja kehitysympäristönä toimi paikallinen Apache-palvelin, jolla pyöritettiin kehitettävää WordPress-sivustoa. Kaikki kehitystyö tehtiin Sublime Textillä, koska se oli jo entuudestaan tuttu työkalu ja sen kanssa toimiminen oli todettu tehokkaaksi.

WordPressissä sivustopohjat rakentuvat PHP:stä ja HTML:stä, eikä näitä kieliä ole varsinaisesti erotettu toisistaan. Tämä aiheuttaa sen, ettei WordPress-sivupohjien kanssa voi käyttää HTML-esikäsittelijää. HTML-esikäsittelijän korvaajaksi löytyi Sublime Textin lisäosa Emmet. Emmetin toiminta ja käyttö poikkeavat HTML-esikäsittelijöistä, mutta hyödyt ovat molemmissa samankaltaiset. Koska Emmet toimii suoraan tekstieditorissa, sen käyttö onnistuu hyvin sivupohjissa jotka sisältävät myös PHP:tä. Emmetin käyttö nopeuttaa HTML-koodin kirjoittamista ja se soveltuu erinomaisesti etenkin kehityksen alkuvaiheeseen kun sivupohjia rakennetaan.

Sivuston tyylitiedostoja varten käyttöön valittiin Sass. Sassin ominaisuudet olivat esitellyistä CSS-esikäsittelijöistä kattavimmat ja sen käyttöönotto oli helppoa. Sass-tiedostot jaoteltiin alla olevan rakenteen mukaisesti (kuva 9). Partials-hakemistossa on sivun rakenteeseen liittyvät tyylitiedostot, sections-hakemiston alta löytyy sivuston eri osa-alueiden tyylitiedostot ja hakemiston juuresta löytyy muuttujia ja apuluokkia varten olevat tyylitiedostot sekä päätyylitiedosto, johon kaikki muut tyylitiedostot ovat sisällytettyinä.

Kuva 9. Sass-tiedostojen jaottelu

Kehityksen tukena käytettiin useita kirjastoja ja lisäosia, joita hallittiin Bowerin avulla. Koska projektissa hyödynnettiin kirjastoja tehokkaasti, ei sivusto vaatinut merkittävästi JavaScript-kehitystä, ja tästä syystä JavaScript-esikäsittelijälle ei ollut tarvetta.

Projektiin otettiin käyttöön myös Grunt-tehtävänsuorittaja, se konfiguroitiin seuraamaan muutoksia Sass- ja main.js-tiedostoissa. Kun Sass-tiedostoihin tulee Gruntin käynnissä ollessa muutoksia, se kääntää automaattisesti Sass-tiedostot uudeksi CSS-tiedostoksi, näin muutokset ovat nopeasti nähtävissä paikallisessa kehitysympäristössä. Ilman automaatiota käännöksen joutuisi tekemään joka kerta manuaalisesti, ennen kuin muutokset tulisivat näkyviin. JavaScript-tiedostoon tehdyt muutokset käynnistävät vastaavanlaisen tehtäväketjun, siinä sivustolla käytetyt lisäosat lisätään main.js-tiedostoon ja tämän jälkeen se minimoidaan.

Projektin versionhallintatyökaluna käytettiin Gitiä. BitBucketin tarjoama ilmainen etävarasto oli yksi merkittävä tekijä, jonka takia Git valittiin. Git oli myös entuudestaan tuttu työkalu, eikä sen toiminnassa ole ollut moittimista. Gitin valintaa puolsi myös, se että satuin löytämään Daniel Koskisen kehittämän.gitignore-tiedoston, joka soveltuu erinomaisesti WordPress-kehitykseen (Koskinen 2013). Jouduin muokkaamaan tiedostoa niin, että se soveltui myös Bowerin ja Gruntin käyttöön (liite 1). Vaikka projektissa oli vain yksi kehittäjä, tein kehitystä muutamalla eri tietokoneella, ja Gitin avulla pystyin pitämään eri tietokoneiden välisen kehityksen helposti ajan tasalla. Git toimi myös projektin varmuuskopiona.

5.3 Tulos

Uusien työkalujen ja -menetelmien käyttöönotto todennäköisesti hidasti kehitystyötä tämän projektin osalta, mutta tässä tapauksessa uusien asioiden oppiminen ja sisäistäminen oli vähintään yhtä tärkeässä roolissa kuin kehitystyön tehostaminen. Tämä kertoo siitä, että ensimmäistä kertaa uusia työkaluja käyttäessä ei välttämättä saada etua vanhoihin kehitystapoihin verrattuna.

Tässä opinnäytetyössä esitellyt työkalut ja -menetelmät tarjoavat hyvän kehityspohjan luomisen. Paras hyöty saadaan silloin, kun kehitetään yleiskäyttöinen projektipohja, joka hyödyntää näitä tarjolla olevia menetelmiä, ja jota voidaan hyödyntää uusissa vastaavanlaisissa projekteissa. Sassin tarjoamat ominaisuudet mahdollistivat responsiivisen sivuston kehittämisen huomattavasti tavallista CSS:ää tehokkaammin. Sisäkkäisten sääntöjen kirjoittaminen mahdollisti sen, että mediakyselyt voitiin sisällyttää valitsimien sisällä ja näin ollen kaikki tiettyyn valitsimeen liittyvät eri mediakyselyt löytyivät samasta paikasta.

Työkalujen ansiosta lopputuloksena saatiin niin sanotusti itsensä dokumentoiva sivusto, jonka kehitystapa ja -käytännöt selviävät helposti käytettyjen työkalujen konfiguraatiosta. Edellytyksenä on kuitenkin se, että kyseiset työkalut ovat kehittäjälle tuttuja. Projekti on uusien työkalujen takia selkeämpi kokonaisuus ja sen jatkokehitys on helpompaa.

Projektin tuloksena saavutettiin valmis WordPress-teema (liite 2) ja tämän lisäksi, konfiguroitu kehityspaketti, joka sisältää WordPress-projektille räätälöidyn Git-versionhallinnan, automatisoidun Sass-kääntämisen sekä Bowerilla hallittujen jQuery lisäosien yhdistämisen yhdeksi tiedostoksi sivustoa varten. Tämän lisäksi tuloksena saadut CSS- ja JavaScript-tiedostot voidaan minimoida tuotantopalvelinta varten, jotta tiedostoille saavutetaan mahdollisimman pieni koko ja tämän myötä nopeampi sivun latausaika

6 Pohdinta

Vaikka opinnäytetyön aihe muuttui tekemisen aikana sivustoprojektista sivuston kehitysmenetelmien tehostamiseen, oli siirtymä luonnollinen, koska aiheet kulkevat vahvasti käsi kädessä. Front end -kehitys on kasvattanut viime vuosina huimaa vauhtia ja se näkyy etenkin uusien kehitystyökalujen ja menetelmien määrässä. Automaatio JavaScript-tehtävänsuorittajilla sekä esikäsittelijöiden hyödyntäminen ovat tulleet jäädäkseen, näiden tueksi on myös kehitetty erilaisia ohjeistuksia parhaita käytäntöjä varten.

Erilaisia päätelaitteita tulee jatkuvasti lisää, ja niiden mukana erilaisia resoluutioita ja näyttökokoja ja tämän lisäksi alan kova kilpailu, ovat tekijöitä, jotka edellyttävät kehitystyökalujen hyödyntämistä. Ilman tehokasta kehitysprosessia ei ole mahdollista tarjota kehitystyötä kilpailukykyiseen hintaan.

Front end -kehitystä tehostavien menetelmien ja työkalujen käyttöönotto voi olla varsinkin isoille yrityksille haastavaa. Menetelmien haltuunotto edellyttää hallittua muutosjohtamista sekä mahdollisesti työntekijöiden lisäkoulutusta. Jotkin uusista teknologioista vanhenevat yhtä nopeasti kuin syntyivät, ja tämä on yksi suuri riskitekijä uusiin työtapoihin siirryttäessä. Uuteen teknologiaan siirtymisestä aiheutuvat kulut voivat näin olla hyötyjä suuremmat. On huomattavasti helpompaa siirtyä käyttämään uusia teknologioita muutaman hengen pienyrityksessä, koska kehitystiimin toiminta on tällöin ketterämpää eikä yritykseltä vaadittava panos ole niin merkittävä. Yleisesti ottaen pienemmän yrityksen projektit ovat nopeampia ja pienempiä, jolloin ne soveltuvat hyvin uusien teknologioiden kokeiluun.

Front end -kehityksen nopean kasvun takia yksittäiset kehittäjät ovat osaltaan vastuussa yrityksen kehitysmenetelmien kehitysnopeudesta ja uusien teknologioiden käyttöönotossa. Vain kehittäjien oman aktiivisuuden kautta voidaan löytää aidosti tehokkaampia työskentelytapoja ja työkaluja.

Kehitystä voidaan tehostaa useilla eri tavoilla, ja on usein projektikohtaista ja tiimikohtaista, millä työkaluilla ja minkälaisilla työtavoilla saadaan suurin hyöty irti. On selvää, ettei pelkkä työkalujen ja uusien menetelmien käyttöönotto takaa tehokasta kehitystä, vaan niitä täytyy osata käyttää myös oikein. Mikäli työkaluja käytetään väärin tai niitä ei osata käyttää kunnolla ne voivat hidastaa kehitystä tehostamisen sijaan.

Front end -kehityksen tehostaminen edellyttää suunnitelmallisuutta. Kun projektin kehitysprosessi on suunniteltu etukäteen hyvin, vältytään monilta ongelmilta ja kehitys sujuu

luontevasti. Silloin myös valitut työkalut ja kehitysmenetelmät ovat todennäköisesti projektiin hyvin sopivat.

Opinnäytetyön eteneminen meni alusta asti kehitys edellä, mutta kehityksen yhteydessä tehtyjen muistiinpanojen avulla kirjoittaminen oli sujuvaa. Suurena haasteena oli aikataulujen sovittaminen kokopäivätyön yhteyteen. Tämä oli jo ennen opinnäytetyötä tiedostettu tekijä, mutta se osoittautui odotettua haastavammaksi opinnäytetyöprosessin alettua. Opinnäytetyön myötä sain syvennettyä front end -kehityksen tietämystä. Entiseen kehitystyönkulkuuni nähden opinnäytetyön aikana opitut menetelmät ovat mullistaneet työmenetelmäni. Opinnäytetyössä esitellyistä esikäsittelijöistä, front end -pakettienhallinnasta sekä automatisoinnista on jokaisesta tullut vakituinen osa uutta työnkulkuani. Nämä uudet työkalut mahdollistavat tehokkaamman kehityksen sekä selkeämmän projektinhallinnan.

Lähteet

Asay, M. 21.1.2014. Git Is Giving Subversion A Run For Its Money: What Took So Long? Luettavissa: http://readwrite.com/2014/01/21/git-subversion-developers. Luettu: 6.11.2014

Bower a. Luettavissa: http://bower.io/. Luettu: 23.10.2014

Bower b. About. Luettavissa http://bower.io/docs/about/. Luettu: 29.10.2014

Bradley, J. 9.6.2014. Using Bower To Improve WordPress Development. Luettavissa: http://code.tutsplus.com/tutorials/using-bower-to-improve-wordpress-development--cms-20524. Luettu: 23.10.2014

Buckler, C. 3.9.2014. SitePoint Smackdown: Atom vs Brackets vs Light Table vs Sublime Text. Luettavissa: http://www.sitepoint.com/sitepoint-smackdown-atom-vs-brackets-vs-light-table-vs-sublime-text/. Luettu: 4.11.2014

Cederholm, D. 13.11.2013. Why Sass? Luettavissa: http://alistapart.com/article/why-sass. Luettu: 24.10.2014

Chouhan, H. 10.6.2014. 10 Compiling Tools for LESS and SASS. Luettavissa: https://www.ostraining.com/blog/coding/compiling-tools/. Luettu: 6.11.2014.

Giraudel, H. 27.2.2014 Architecture for a Sass Project. Luettavissa: http://www.sitepoint.com/architecture-sass-project/. Luettu: 18.11.2014

Git. Getting Started - A Short History of Git. Luettavissa: http://git-scm.com/book/en/Getting-Started-A-Short-History-of-Git. Luettu: 24.10.2014

Gyllenswärd, M. 14.11.2013. Sass Variables. Luettavissa: http://robots.thoughtbot.com/sass-variables. Luettu: 18.11.2014

Hansson, A. 28.2.2012. The Basics of jQuery. Luettavissa: http://andreehansson.se/the-basics-of-jquery/. Luettu: 19.11.2014.

Howe, S. Preprocessors. Luettavissa: http://learn.shayhowe.com/advanced-html-css/preprocessors/. Luettu: 22.10.2014

Jenson, G. 12.3.2014. Why should you use CoffeeScript instead of JavaScript? Luettavissa: http://www.maori.geek.nz/post/why_should_you_use_coffeescript_instead_of_javascript. Luettu: 6.11.2014

Johnson, J. 13.2.2012. Save Loads of Time by Writing Your HTML With Haml. Luettavissa: http://designshack.net/articles/html/save-loads-of-time-by-writing-your-html-with-haml/. Luettu: 18.11.2014

Koskinen, D. 2.4.2013. My Dev Setup, part 4: Version Control and Deployments with WordPress. Luettavissa: http://danielkoskinen.com/version-control-and-deployments/. Luettu: 11.11.2014

Osmani, A. 30.6.2013. Checking in front-end dependencies. Luettavissa: http://addyosmani.com/blog/checking-in-front-end-dependencies/. Luettu: 29.10.2014

Park, T. 12.3.2013. Customizing Bootstrap. Luettavissa: http://www.smashingmagazine.com/2013/03/12/customizing-bootstrap/. Luettu: 18.11.2014

Patton, T. 21.10.2014. Atom text editor provides flexible options for working with various file formats. Luettavissa: http://www.techrepublic.com/article/atom-text-editor-provides-flexible-options-for-working-with-various-file-formats/. Luettu: 18.11.2014

Pettit, N. 2.12.2013. Which Text Editor Should I Use? Luettavissa: http://blog.teamtreehouse.com/which-text-editor-should-i-use. Luettu: 28.10.2014

Shull, E. 9.4.2014. Graduating from CoffeeScript to LiveScript. Luettavissa: http://spin.atomicobject.com/2014/04/09/coffeescript-livescript/. Luettu: 6.11.2014

Symfony. Why should I use a framework? Luettavissa: http://symfony.com/why-use-a-framework. Luettu: 22.10.2014

Stansberry, G. 18.9.2008. 7 Version Control Systems Reviewed. Luettavissa: http://www.smashingmagazine.com/2008/09/18/the-top-7-open-source-version-control-systems/. Luettu: 24.10.2014

Tucker, D. 12.10.2013. Automating The Development, Build, and Deployment Process with Grunt. Luettavissa: http://davidtucker.net/articles/automating-with-grunt/. Luettu: 23.10.2014

Wendell, M. 9.4.2012. Responsive Web Design in Sass: Using media queries in Sass 3.2. Luettavissa: http://thesassway.com/intermediate/responsive-web-design-in-sass-using-media-queries-in-sass-32. Luettu: 18.11.2014

West, M. 30.12.2013. Getting Started with Bower. Luettavissa: http://blog.teamtreehouse.com/getting-started-bower. Luettu 29.10.2014

Zhitnitsky, A. 13.8.2014. Sublime VS. Atom: Can GitHub Take the Lead? Luettavissa: http://www.takipiblog.com/sublime-vs-atom-text-editor-battles/. Luettu: 18.11.2014

Liitteet

Liite 1. Projektin .gitignore-tiedoston sisältö

```
# This is a template .gitignore file for git-managed WordPress projects.
#
# Fact: you don't want WordPress core files, or your server-specific
# configuration files etc., in your project's repository. You just don't.
#
# Solution: stick this file up your repository root (which it assumes is
# also the WordPress root directory) and add exceptions for any plugins,
# themes, and other directories that should be under version control.
#
# See the comments below for more info on how to add exceptions for your
# content. Or see git's documentation for more info on .gitignore files:
# http://kernel.org/pub/software/scm/git/docs/gitignore.html

# Ignore everything in the root except the "wp-content" directory.
/*
!.gitignore
!wp-content/

# Ignore everything in the "wp-content" directory, except the "plugins"
# and "themes" directories.
wp-content/*
!wp-content/plugins/
!wp-content/themes/

# Ignore everything in the "plugins" directory, except the plugins you
# specify (see the commented-out examples for hints on how to do this.)
wp-content/plugins/*
# !wp-content/plugins/my-single-file-plugin.php
# !wp-content/plugins/my-directory-plugin/

# Ignore everything in the "themes" directory, except the themes you
# specify (see the commented-out example for a hint on how to do this.)
wp-content/themes/*
!wp-content/themes/audile/
```

```
# Ignore "tmp" directory in "themes/audile/"
wp-content/themes/audile/tmp/

# Ignore sass-cache
wp-content/themes/audile/.sass-cache/

# Ignore node_modules/ but do not ignore bower_components (using non-registered packages)
wp-content/themes/audile/node_modules/
!wp-content/themes/audile/bower_components/
```

Liite 2. Sivuston lähdekoodi (salassapidettävä)